Δομές Δεδομένων-Εργασία 1

Πλαβός Δημοσθένης 3200156

ΜΕΡΟΣ Α

Σημείωση: Χρησιμοποιώ βοηθητικά την κλάση Node της οποίας τα αντικείμενα αντιπροσωπεύουν τους κόμβους που χρησιμοποιώ για την αποθήκευση στοιχείων στη στοίβα και στην ουρά με το πεδίο data να περιέχει την τιμή του κόμβου και το πεδίο next να περιέχει τη διεύθυνση του επόμενου στη σειρά κόμβου.

Στο Μέρος Α, ζητείται η κατασκευή 2 κλάσεων, της StringStackImpl (υλοποίηση στοίβας) και της StringQueueImpl (υλοποίηση ουράς).

1) Όσον αφορά στη στοίβα, δημιούργησα την κλάση StringStackImpl που χρησιμοποιεί μία λίστα μονής σύνδεσης για την αποθήκευση των στοιχείων της στοίβας, και η οποία υλοποιεί τη διεπαφή StringStack.

Πεδία της StringStackImpl:

- head : δείχνει στον πρώτο κόμβο της στοίβας
- size: μέγεθος της στοίβας

Παράλληλα, έκανα χρήση generics, ώστε να μπορώ να χρησιμοποιήσω οποιονδήποτε τύπο δεδομένων για την εισαγωγή τους στη στοίβα.

Στην κλάση ανέπτυξα τις παρακάτω μεθόδους:

- ❖ isEmpty() : ελέγχει αν η στοίβα είναι άδεια. Επιστρέφει true αν είναι, αν όχι επιστέφει false.
- ❖ push (T item) : εισάγει ένα αντικείμενο τύπου T στη στοίβα.
- ❖ pop() : αφαιρεί το αντικείμενο που εισήχθη τελευταίο στη στοίβα (head). Αν η στοίβα είναι άδεια, το πρόγραμμα πετάει εξαίρεση NoSuchElementException.
- ❖ peek() : επιστρέφει αλλά δεν αφαιρεί το head. Αν η στοίβα είναι άδεια, το πρόγραμμα πετάει εξαίρεση NoSuchElementException.
- ❖ printStack (PrintStream stream) : εκτυπώνει την στοίβα ξεκινώντας από το head, δηλαδή από το τελευταίο εισαχθέν στοιχείο, ενώ αν είναι άδεια, τυπώνει ανάλογο μήνυμα.
- ❖ size() : επιστρέφει το μέγεθος της στοίβας.

Πλαβός Δημοσθένης (ΑΜ:3200156)

Αντίστοιχα, ζητείται η κατασκευή μίας κλάσης StringQueueImpl που χρησιμοποιεί λίστα μονής σύνδεσης για την υλοποίηση ουράς FIFO, και η οποία κλάση υλοποιεί τη διεπαφή StringQueue.

Κάνω χρήση generics έτσι ώστε να μπορώ να χρησιμοποιήσω οποιονδήποτε τύπο δεδομένων επιθυμώ.

Πεδία της StringQueueImpl:

- tail: δείχνει στον κόμβο της ουράς που εισήχθη νεότερα,
- head: δείχνει στον κόμβο της ουράς που εισήχθη παλαιότερα
- size: μέγεθος της ουράς.

Σχετικά με τις μεθόδους:

Χρησιμοποιούνται ομοίως με τη στοίβα οι: isEmpty() , size() , peek()

Επιπλέον οι:

- ❖ put (T item) : προσθέτει ένα αντικείμενο τύπου T στην ουρά.
- ❖ get() : αφαιρεί τον πρώτο κόμβο της ουράς (head). Αν η ουρά είναι άδεια, το πρόγραμμα πετάει εξαίρεση τύπου NoSuchElementException.
- ❖ printQueue(PrintStream stream) : εκτυπώνει την ουρά από την αρχή (head) προς το τέλος (tail) , ενώ αν είναι άδεια, τυπώνει ανάλογο μήνυμα.

ΜΕΡΟΣ Β

Στο μέρος Β δημιουργώ την κλάση Thiseas.java, όπου πρέπει να κάνω χρήση της στοίβας , προκειμένου να βρω την έξοδο του λαβυρίνθου.

Τα βήματα της υλοποίησης είναι τα εξής:

Χρησιμοποιώ την BufferedReader για να αναγνώσω το αρχείο. Κάνω ένα πέρασμα όλο το αρχείο γραμμή-γραμμή μέχρις ότου να μην υπάρχει κείμενο και χρησιμοποιώ:

- τις μεταβλητές width, height, coordinateX, coordinateY για να περάσω το

πλάτος, το ύψος και τις συντεταγμένες του σημείου Ε (σημείο εκκίνησης) αντίστοιχα

- το δισδιάστατο array labyrinth για να περάσω τις τιμές που περιέχει ο λαβύρινθος σε κάθε σημείο του, χρησιμοποιώντας τις συντεταγμένες του λαβυρίνθου (height, width).

Αν τα δεδομένα που υπάρχουν στο κείμενο δεν είναι τα επιθυμητά, πετάω εξαίρεση, πχ αν οι τιμές των 2 πρώτων γραμμών δεν είναι 2 σε πλήθος ή αν δεν είναι αριθμητικές τιμές. Για το τελευταίο χρησιμοποιώ τη συνάρτηση isNumeric που παίρνει σαν όρισμα ένα String και πετάει εξαίρεση αν η μετατροπή του σε αριθμό δεν είναι εφικτή. Η μέθοδος αυτή είναι στατική για να μπορώ να τη καλέσω και χωρίς αντικείμενο.

Στο array labyrinth περνάω γύρω γύρω τιμές «Χ», οι οποίες σημαίνουν ότι φτάσαμε σε έξοδο. Επίσης, αν η τιμή Ε(εκκίνηση) βρίσκεται σε κάποιο άκρο το πίνακα (πρώτη,τελευταία στήλη ή πρώτη,τελευταία γραμμή) θεωρώ ότι δεν μπορεί η έξοδος να βρίσκεται επίσης εκεί, οπότε περνάω τιμές «Α».

Πχ.

1 1 1 Ε **1 0 1**

1 0 0 0 1 0 1

1 0 1 0 1 0 1

1 0 1 0 0 0 1

1 1 1 0 1 1 1

1 0 0 0 0 0 1

1 0 1 1 1 0 1

1 0 1 1 0 0 1

0 1 1 1 1 0 1

Οι τιμές σε Bolt δεν είναι αποδεκτές για να μην μπορεί πχ ο παίκτης να βρει την έξοδο ακριβώς δίπλα του.

Αρχικά, δημιουργώ αντικείμενο της κλάσης Thiseas με όνομα“shmeio” για να χρησιμοποιώ τις αντίστοιχες μεθόδους του getX(), getY(), getValue() κλπ.

Στη συνέχεια, έχοντας την τελική μορφή του array , το χρησιμοποιώ για να κάνω διάσχιση του λαβυρίνθου. Όπου βρίσκω τιμή 0, θέλω να συνεχίζω. Κινούμαι με προτεραιότητα στα αριστερά, μετά κάτω, μετά στα δεξιά, και τέλος πάνω.

Με τη δεδομένη προτεραιότητα κινήσεων, όπου βρω τιμή 0 την μετατρέπω:

- είτε σε «F» το οποίο σημαίνει ότι έχω διασχίσει το συγκεκριμένο σημείο του λαβυρίνθου
- είτε σε «K» το οποίο σημαίνει ότι το σημείο αυτό είναι σημείο ΚΟΜΒΟΣ, δηλαδή σημείο που έχει 2 και πάνω επιλογές κατεύθυνσης (πχ μπορείς μετά από αυτό να βρεις 0 και αριστερά και κάτω). Το σημείο αυτό χρησιμοποιώ έτσι ώστε όταν βρίσκω αδιέξοδο να επιστρέφω σε αυτό και όχι πάλι στην αρχή του λαβυρίνθου.

  Κάθε φορά που διασχίζω ένα σημείο του λαβυρίνθου και το μετατρέπω στον αντίστοιχο χαρακτήρα, περνάω τις συντεταγμένες του στη στοίβα με όνομα “stoiva” και όποτε βρίσκω αδιέξοδο (δηλαδή δεν έχω τιμή 0 προς καμία κατεύθυνση) επιστρέφω στα σημεία Κ και αδειάζω τη στοίβα αντιστρόφως μέχρι να φτάσω στο Κ.

  Αν φτάσω σε σημείο εξόδου, εμφανίζω μήνυμα με τις συντεταγμένες του ενώ αντιθέτως αν τελειώσουν τα σημεία Κ και η αναζήτηση των πιθανών μονοπατιών , εμφανίζω μήνυμα ότι δεν υπάρχει έξοδος απ’ το λαβύρινθο.

ΜΕΡΟΣ Γ

Στο μέρος Γ έχω φτιάξει την κλάση StringQueueWithOnePointer η οποία είναι μία υλοποίηση ουράς FIFO με τη χρήση όμως ενός δείκτη, Επιλέγω να κάνω χρήση του πεδίου tail το οποίο δείχνει στον νεότερο κόμβο της ουράς. Αυτό επιτυγχάνεται με τη χρήση κυκλικής λίστας, όπως θα εξηγήσω παρακάτω.

Αρχικά, αναφέρω ότι το tail.next είναι αντίστοιχο με το head που έχουμε στην υλοποίηση της ουράς με τους δείκτες head, tail.

Μέθοδοι: Στην κλάση ανέπτυξα τις παρακάτω μεθόδους:

- isEmpty() : ελέγχει αν η ουρά είναι άδεια. Επιστρέφει true αν το tail δείχνει σε null κόμβο, αν όχι επιστέφει false.
- put (T item) : εισάγει ένα αντικείμενο τύπου T στην ουρά. Αν η ουρά είναι άδεια, τότε και το tail και το tail.next δείχνουν στον κόμβο που εισάγεται, μιας και είναι ο μοναδικός. Αν όχι, τότε έχοντας δημιουργήσει το νέο αντικείμενο(κόμβος) και το temp δείχνει σε αυτό, χρησιμοποιώ το temp ως εξής: Το temp.next δείχνει στο tail.next που είναι ο επόμενος κόμβος του tail. Έπειτα, το tail.next δείχνει στον νέο κόμβο (στον οποίο δείχνει ο temp) άρα ο νέος μπαίνει μετά το tail.next. Τέλος, το tail δείχνει στον νέο κόμβο, ο οποίος βρίσκεται στο τέλος της ουράς και το tail εξακολουθεί να δείχνει στον νεότερο κόμβο όπως πρέπει.
- pop() : αφαιρεί το αντικείμενο που εισήχθη παλαιότερα στην ουρά. Αν η ουρά έχει μόνο έναν κόμβο, τότε το tail δείχνει σε null. Αν έχει παραπάνω από έναν κόμβο, τότε κρατάμε την τιμή του κόμβου στη μεταβλητή returnedValue (για να την κάνουμε return) και, το tail.next (το οποίο δείχνει στο παλιότερο στοιχείο της ουράς που πρέπει να φύγει) δείχνει στον επόμενο κόμβο, δηλαδή στο tail.next.next.
- peek() : επιστρέφει την τιμή που περίεχει ο παλαιότερος κόμβος της ουράς, δηλαδή το tail.next.
- printStack (PrintStream stream) : εκτυπώνει την ουρά ξεκινώντας από το παλιότερο στοιχείο μέχρι το νεότερο. Η συνθήκη τερματισμού είναι “tail<>temp”, δηλαδή όταν μετά από το πέρασμα της ουράς φτάσω πάλι στο tail, σταματάω.
- size() : επιστρέφει το μέγεθος της ουράς.